AD-A047 353    MARYLAND UNIV COLLEGE PARK COMPUTER SCIENCE CENTER    F/G 9/2
DESIGN OF AN ADAPTIVE, PARALLEL FINITE ELEMENT SYSTEM.(U)
NOV 77  P ZAVE, W C RHEINBOLDT    N00014-77-C-0623
UNCLASSIFIED    TR-593    NL

| OF |
AD
A047353

END
DATE
FILMED
1 -78
DDC

12
B.S.

# COMPUTER SCIENCE TECHNICAL REPORT SERIES

UNIVERSITY OF MARYLAND 1807-1856-1920

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

[illegible]

12

Technical Report TR-593
ONR-N00014-77-C-0623-593

November 1977

# Design of an Adaptive, Parallel Finite Element System[1)]

Pamela Zave[2)] and Werner C. Rheinboldt[3)]

(See 1473)

1)This research was supported in part by the Office of Naval Research, Mathematics Branch, under Contract N00014-77-C-0623.

2)Computer Science Department, University of Maryland, College Park, MD

3)Department of Mathematics and Computer Science Center, University of Maryland, College Park, MD

## Abstract

A design for a novel prototype finite element system is presented which meets the following four goals: (a) The system constitutes an application-independent finite element solver for a certain class of linear elliptic problems based on a weak mathematical formulation; (b) it incorporates extensive adaptive approaches to minimize the critical decisions demanded of the user; (c) it incorporates adaptive error estimation techniques to provide an optimal solution within a prescribed cost range; and (last but not least) (d) it takes advantage of natural parallelism and modularity to increase the size of the practically solvable problems. The overall system structure is described in terms of parallel processes and is implementable on a wide variety of hardware configurations. The experiments planned to evaluate the various new approaches are also presented.

# DESIGN OF AN ADAPTIVE, PARALLEL FINITE ELEMENT SYSTEM

Pamela Zave and Werner C. Rheinboldt

## 1. Introduction

In the past two decades the finite element method has found increasingly widespread application in nearly all areas of engineering and science. Many, often large, software systems based on the method have been developed and are used extensively (see, e.g., [1]). However, most of them have a relatively inflexible program structure and demand of the users numerous critical and frequently difficult decisions.

At the same time recent years have brought considerable advances in the theoretical analysis of the method and in the development of efficient solution procedures as well as in the understanding of modern software systems. There appears to be a considerable need for utilizing these and related advances in the design of the next generation of finite element software.

Various authors have summarized current trends in the design of the next generation of finite element programs (see, e.g., [2],[3]). The following idealized design criteria may be somewhat more radical but appear to us highly desirable and yet in the realm of practicality:

(1) The programs should be designed not so much for specific classes of applications, but instead should constitute more general "finite element solvers" of entire classes of partial differential equations. This will ensure a much broader applicability and provide also for a closer interaction with the theoretical foundations.

(2) The programs should incorporate extensive application of adaptive solution methods in order to relieve the user of the burden of many of the critical a-priori decisions he now has to make.

(3) The overall process should produce a solution which meets optimal accuracy requirements within a prescribed range of computational cost. Hence the design should include well thought-out provisions for the assessment of the relevant errors and computational costs. This is also essential for the desired adaptive control of the course of the computation.

(4) The system should be designed to take advantage of any natural parallelism and modularity of the finite element method. This represents practically the only open avenue toward gaining significant improvements in overall solution time or in the size of the solvable problems.

Clearly these design criteria are still ideals, and much work will be needed before they can be fully realized. However, as mentioned before, they are within the realm of practicality.

In substantiation of this claim we present here the design of an experimental finite element system that incorporates the principal aspects of all four criteria. The design is based on earlier work ([4]-[9]) about the questions behind (1), (2), and (3), as well as on recent results ([10]-[12]) about parallel process structures and their specification.

The introduction of parallelism appears to be a particularly novel aspect of the design. This parallelism is on the procedural level rather than the instruction level because there the expected payoff is much greater. We note that execution of the numerical algorithms by parallel programs for vector processors is neither included in, nor precluded by, the design.

Parallelism is specified in terms of processes which are autonomous units with their own programs and data. Processes run in parallel and communicate asynchronously in a limited and highly structured manner.

In the design the local data of a process contain almost all the information needed by the functions computed in that process. If this were not so, the overhead and delays of interprocess communications would readily smother the expected parallelism. Thus the process structure induces on the finite element data a segmentation which is rather natural to the problem and can be used as the basis for intelligent storage management in the environment of a single large computer. Data segmentation may prove as important as parallelism in expanding the size of the solvable problems. In fact, it enables our system to perform automatically the problem segmentation that is now carried out at the user's level.

The use of parallel processes in the design makes it possible to apply multiple processors effectively, hopefully for significant gains in speed (we conjecture that the fastest possible implementation of this approach to the finite element method would be on a network of small computers with communication links to fit our structure). This does not limit the design to such specialized architectures, however; the operating system of a typical large computer simulates parallel processes, and the data segmentation mentioned above should provide significant advantages.

In Chapter 2 below we present the theoretical background for the finite element computations incorporated in the system. Chapter 3 describes the systems design itself, and Chapter 4 introduces the experimental system now under construction and summarizes the performance evaluation mechanisms that will be included in it.

## 2. The Mathematical Basis of the Design

By necessity our attention had to be restricted to a specific class of problems that is sufficiently broad to allow for meaningful conclusions and yet narrow enough for easy implementation as an experimental system. Our choice was a fairly general class of linear, stationary, elliptic boundary value problems on certain two-dimensional domains. At the same time, throughout the design we refrained from using approaches that would not admit extensions to more general problems.

In line with the first design criterion the system is based on a weak formulation of the given problem in terms of an appropriate bilinear form $B$ on certain Hilbert spaces $H_1, H_2$ and a corresponding (load) functional $f$ on $H_2$. In other words the underlying mathematical problem is to obtain the (unique) $u_0 \in H_1$ such that

$$B(u_0, v) = f(v), \ \forall\, v \in H_2 \ . \tag{2.1}$$

The finite element method then proceeds to the construction of suitable, finite-dimensional subspaces $M_1 \subset H_1$, $M_2 \subset H_2$ and the numerical computation of the solution $\hat{u}_0 \in H_1$ of

$$B(\hat{u}_0, v) = f(v), \ \forall\, v \in M_2 \ . \tag{2.2}$$

The choice of $B$ and $f$ determines the problem and the structure of the finite element method. The selection of the spaces $H_1, H_2$ affects the norms used in the error analysis and hence the adaptive control of the overall solution process. Finally the spaces $M_1, M_2$ derive from the selection of the specific finite elements; that is, their definition is equivalent to the construction of a particular finite element mesh on the given domain $\bar{\Omega} \subset R^2$. Our use of adaptive mesh refinement, of course, involves consideration of a sequence of subspace pairs $M_1, M_2$ during the course of the overall solution.

We refer to [13] for the theoretical background of the above formulation and to [ 7 ] for some of the reasoning behind its use as a basis for our system design.

Many practical finite element applications employ a so-called substructure analysis (see, e.g. [14] where other references are also given). Broadly speaking the domain $\bar{\Omega}$ is defined as a union of subdomains and on each one of them a finite element mesh is introduced. The solution then proceeds in levels; that is, all degrees of freedom internal to each subdomain are eliminated first and then, with the substructures acting as "superelements", the full solution is obtained.

This technique suggests a natural parallel process structure for the finite element system. In fact, to a considerable extent the computations on the different substructures are independent of each other. Hence we may consider associating each one of them with one (or several) individual processors which in turn operate essentially in parallel. This basic design concept will be developed further in subsequent sections. We observe that the subdomain segmentation of the data and processing is the same as that used in large industrial applications,except that ours is automatic and flexible,while current finite element systems demand that their users create a rigid segmentation at the manual or even managerial level!

For the domain $\bar{\Omega} \subset R^2$ we are led to assume that $\bar{\Omega}$ is the union

(2.3) $$\bar{\Omega} = \bar{\Omega}_1 \cup \bar{\Omega}_2 \cup \ldots \cup \bar{\Omega}_N$$

of finitely many closed, bounded subsets $\bar{\Omega}_i \subset R^2$ which have nonempty interiors $\Omega_i$ such that

(2.4) $$\Omega_i \cap \Omega_j = \emptyset, \; i \neq j \; .$$

For the design of a reasonably efficient mesh refinement algorithm, further restrictions on the choice of the subdomains are desirable. We assume here that each $\bar{\Omega}_i$ is a diffeomorphic image of some fundamental figure in $R^2$ on which a simple hierarchy of subdivisions can be defined. For simplicity of the implementation, the experimental system employs only one fundamental figure, namely, the unit square

(2.5) $$\bar{Q} = \{\xi,\eta \in R^2 \mid 0 \leq \xi, \; \eta \leq 1\} \; .$$

The use of other figures does not introduce any particular difficulties and, with appropriate interface provisions, different fundamental sets may be used to define the various subdivisions.

We call $\bar{Q}$ and its interior $Q$ the closed and open base square. The corners of $\bar{Q}$ are denoted by $\text{point}_i(Q)$, $i = 1,2,3,4$, the closed sides by $\overline{\text{side}}_i(Q)$ and their relative interiors by $\text{side}_i(Q)$, $i = 1,2,3,4$.

For each subdomain $\bar{\Omega}_i$ a diffeomorphism $\psi_i$ is given that maps $\bar{Q}$ onto $\bar{\Omega}_i$. Moreover, we assume that for any $i \neq j$ exactly one of the following conditions holds:

(i) $\bar{\Omega}_i \cap \bar{\Omega}_j = \emptyset$

(ii) $\bar{\Omega}_i \cap \bar{\Omega}_j = \psi_i(\text{point}_{\ell_i}(Q)) = \psi_j(\text{point}_{\ell_j}(Q))$

(2.6) (iii) $\bar{\Omega}_i \cap \bar{\Omega}_j = \psi_i(\overline{\text{side}}_{\ell_i}(Q)) = \psi_j(\overline{\text{side}}_{\ell_j}(Q))$, and the restriction of $\psi_i$ to $\overline{\text{side}}_{\ell_i}(Q)$ is identical with the restriction of $\psi_j$ to $\overline{\text{side}}_{\ell_j}(Q)$.

The open subdomains $\Omega_i$ shall be called 2-subdomains. By (2.4) it is meaningful to speak of the open and closed sides of $\Omega_i$, namely, the sets $\text{side}_\ell(\Omega_i) = \psi_i(\text{side}_\ell(Q))$ and $\overline{\text{side}}_\ell(\Omega_i) = \psi_i(\overline{\text{side}}_\ell(Q))$, $\ell = 1,2,3,4$, respectively. The open sides of $\Omega_i$ are named 1-subdomains. Finally we define the points of $\Omega_i$ by $\text{point}_\ell(\Omega_i) = \psi_i(\text{point}_\ell(Q))$ and call them 0-subdomains.

Each closed side of $\Omega_i$ is the diffeomorphic image of the unit interval $I = \{s \in R^1 \mid 0 \le s \le 1\}$. For the implementation only these one-dimensional mappings are given and the mappings $\psi_i$ are constructed by standard bivariate blending techniques (see, e.g., [15]).

Let $\Omega^{(i)} \subset \Omega$ be the union of the i-subdomains, i = 0,1,2. Then we consider the second order form

$$
\begin{aligned}
B(u,v) = \int_{\Omega} \sum_{i,j=1}^{m} \Bigg[ & \sum_{k,\ell=1}^{2} a_{k,\ell}^{i,j}(x) \frac{\partial u_i}{\partial x_k} \frac{\partial v_j}{\partial x_\ell} \\
& + \sum_{k=1}^{2} \left( b_k^{i,j}(x) \frac{\partial u_i}{\partial x_k} v_j + \hat{b}_k^{i,j}(x) u_i \frac{\partial v_j}{\partial x_k} \right) + c^{i,j}(x) u_i v_j \Bigg] dx \\
(2.7) \qquad + \int_{\Omega^{(1)}} \sum_{i,j=1}^{m+\hat{m}} \Bigg[ & \alpha^{i,j}(s) \frac{\partial u_i}{\partial s} \frac{\partial v_j}{\partial s} + \beta^{i,j}(s) \frac{\partial u_i}{\partial s} v_j \\
& + \hat{\beta}^{i,j}(s) u_i \frac{\partial v_j}{\partial s} + \gamma^{i,j}(s) u_i u_j \Bigg] ds \; .
\end{aligned}
$$

Here we have

$$u = (u_1, \ldots, u_m, u_{m+1}, \ldots, u_{m+\hat{m}})$$

$$v = (v_1, \ldots, v_m, v_{m+1}, \ldots, v_{m+\hat{m}})$$

where the functions $u_1, \ldots, u_m, v_1, \ldots, v_m$ are defined on $\Omega$ and the remaining ones on $\Omega^{(1)} \cup \Omega^{(0)}$.

In general the right side of the equation (2.1) has the same form as (2.7) after $u$ is fixed; but, of course, the coefficients may be different. It should be evident how more general classes of forms may be defined.

The formulation (2.7) is already fairly general. It includes not only most classical problems and finite element methods but also the elliptic problems of Douglis-Nirenberg type, the forms arising in the use of the least squares method for first-order systems, various versions of the Lagrange multiplier methods, and many others.

For the implementation a library of coefficient functions is introduced from which a set of coefficients is chosen for each 2-subdomain and 1-subdomain. It will have to be assumed that B satisfies the conditions required by the theory (see, e.g. [13]) on some pair of Hilbert spaces $H_1, H_2$. As mentioned before, the spaces $H_1, H_2$ only affect the norm used in the error estimates. A library is provided for computing the function norms on the most important spaces.

The mesh on each $\bar{\Omega}_i$ consists of curvilinear elements which are first defined on the basic square Q and then mapped into $\bar{\Omega}_i$. The element selection in Q is restricted by the condition that only those elements are to be used which would arise in regular subdivisions of Q. Hence our choice of the unit square Q as basic figure implies that the elements are defined either on squares of the form

$$(2.8) \qquad q = \{(\xi,\eta) \in Q \mid ih \le \xi \le (i+1)h,\ jh \le \eta \le (j+1)h\},$$

$$h = 2^{-k},\ 0 \le i,j \le 2^k - 1,$$

or on right triangles obtained by subdividing these squares along a fixed diagonal.

In order to simplify the discussion we confine ourselves here to bilinear elements on the squares (2.8). Higher-order Lagrangian or Hermitian elements could be used as well; they only increase somewhat the level of complexity of the programs. It should also be noted that the use of adaptive mesh refinement reduces somewhat the need for high-order elements.

The admissible meshes on Q may be defined recursively by the two rules:

(2.9)
(i) A mesh T on Q is admissible if it consists of all $2^{2k}$ squares (2.8) with a fixed $h = 2^k$, $k \geq 0$.

(ii) If T is an admissible mesh on Q, then the mesh T' is admissible that is obtained from T by subdividing any one single square q of T into four congruent squares of half the side-length of q.

Figure 2.1 shows an example of an admissible mesh on Q.

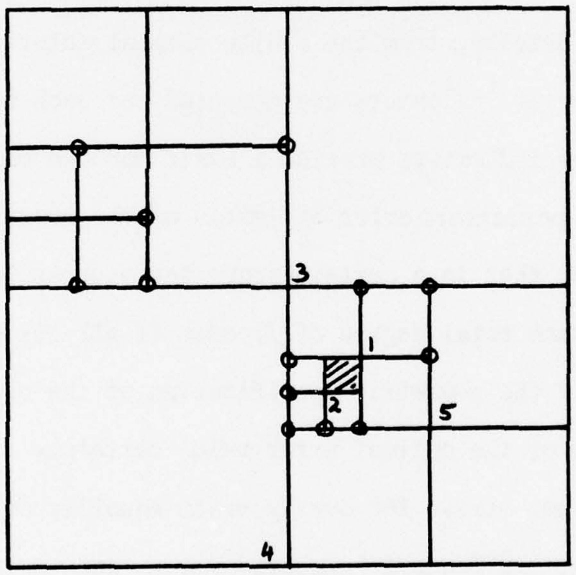

Figure 2.1

In such meshes some elements of different size border on each other. The resulting "illegal" nodes--in Figure 2.1 marked by circles--can be handled in a variety of ways. For simplicity, the values at these illegal nodes are defined here by continuity. For instance, in our case of bilinear elements the elementary stiffness matrix $K$ of the hatched element depends on the five numbered nodes. If $K_0$ represents the usual stiffness matrix of that element, then $K = L^T K_0 L$ where $L$ is an easily computable interpolation matrix. While the legality or illegality of the nodes on the boundary depends on the mesh in $\Omega$, the information needed to compute any $K$ does not.

The mesh refinement procedure is described in [8], and we shall not repeat the details. Briefly, from the finite element solution on some mesh on $\Omega$ certain error indicators are computed for each element. As shown in [8],[9] these indicators provide a basis for the computation of highly realistic a-posteriori error estimates of the overall solution. Moreover, it turns out that in a certain sense the mesh is "optimal" among meshes of the same total degree of freedom if all its error indicators are equal. While the geometric specification of the optimal mesh is not very stable, that of the optimal error value certainly is stable. Accordingly, we need not strive for overly exact equality of the error indicators. For this, two "cutoff" numbers $\varepsilon_{high}$ and $\varepsilon_{low}$ are determined which specify that all elements with indicators above $\varepsilon_{high}$ have to be refined and that those elements with values above $\varepsilon_{low}$ are to be included in the reassembly and subsequent solution of the macro-stiffness matrix. On all other elements the present solution is retained and--where needed--used as a boundary condition.

The process is started with a uniform mesh of the form (2.9i) on all $\Omega_i$, all error indicators set to $\infty$, and $\varepsilon_{high} = \infty$, $\varepsilon_{low} = 0$. It terminates either when the error estimate based on the computed indicators falls below a given tolerance or when a prescribed maximum computational cost is exceeded.

Thus a basic solution pass has the form:

(2.10)

1. Determine the cutoffs $\varepsilon_{high}$ and $\varepsilon_{low}$;
2. Check for termination;
3. Refine all elements with indicators above $\varepsilon_{high}$;
4. Compute the micro-stiffness matrices for all elements with errors above $\varepsilon_{low}$ and assemble the corresponding macro-stiffness matrix;
5. Solve the resulting equations;
6. Compute the error indicators for all elements involved in the solution.

For details we refer to the next section.

In the experimental system standard sparse matrix approaches and elimination techniques are used in the numerical solution of the macro-stiffness equations. It should be noted here that the tree structure of the refinement process leads naturally to a very effective ordering of the equations and unknowns. Alternately iterative procedures can be used throughout the intermediate solution passes. In either case a direct solution on the complete final mesh is strongly indicated. This can be accomplished by performing a final solution pass with $\varepsilon_{high} = \infty$, $\varepsilon_{low} = 0$.

# 3. The Systems Design

## 3.1 Requirements

Besides the design criteria mentioned in the introduction, the system is to incorporate the following user interface features:

(1) A problem is defined by a given domain $\Omega$, bilinear form $B$, and associated error norms. Each instance of the system is initialized with a particular problem. It can accommodate several users working simultaneously on this problem, each one with his own boundary conditions, load functional, sequence of meshes, and solutions.

(2) The design is to provide for a "front-end" with user-oriented features such as

  (a) pre- and post-processing facilities;

  (b) graphics;

  (c) a library of bilinear forms and associated norms, retrievable by classifications associated with the fields of study and the characteristics of the physical problems;

  (d) a command language that can be used at varying levels of sophistication.

In addition, the systems design should meet the following performance goals:

(3) The system must incorporate as much procedural parallelism as possible.

(4) Data are not to be stored redundantly since most data structures in practical finite element systems are large in size.

(5) For effective storage management the data sets should be factored.

(6) The design should be implementable on a variety of computer systems.

## 3.2 The Overall Process Structure

Formally a process is a pair $(\Sigma,f)$ consisting of a state space $\Sigma$, that is, a set of possible values or states of a data structure, and $f : \Sigma \rightarrow \Sigma$ a successor function. Hence a sequence of states $\sigma_1,\sigma_2,\sigma_3,\ldots$ is a computation of a process $(\Sigma,f)$ **if** $\sigma_i \in \Sigma$ and $\sigma_{i+1} = f(\sigma_i)$ for all $i \geq 1$. Sets of asynchronously interacting parallel processes are abstract characterizations of digital systems and hence can be implemented in many ways.

The design discussed here has been formally specified in terms of a language based on the above abstract process concept (see [12]). In this chapter we present only an informal description of the process structure and the interprocess communications. A more detailed, semi-formal presentation in terms of an ALGOL-like language is given in the appendix.

As indicated earlier, the design is based on the idea that parallel activities reside in different processes. In line with the discussion of Chapter 2, the following areas of potential parallelism were identified:

(1) While one user proceeds with a finite element solution, other users can use the "front-end" facilities described in Section 3.1.

(2) Operations that can be carried out in parallel on different parts of the domain include:

(a) adaptive refinement of the mesh;

(b) computation of the micro-stiffness matrices;

(c) computation of the error indicators on each element.

(3) During many solution passes a new solution is computed only on a subset of the existing nodes. If these active nodes fall into different

noncontiguous subdomains, the corresponding linear systems are independent of each other and can be solved in parallel.

Corresponding to this, the design incorporates three principal classes of processes:

(1) The user [illegible]ses, each representing one of the current systems users.

(2) The geom[illegible] corresponding to the 2-, 1-, and 0-subdomains described in [illegible].

(3) The linear systems solvers.

*The state of a user process contains the data particular to that user* and any information needed for post-processing, graphics, etc. In the formal specification of the system, a user process represents both human and automatic components. For example, there is a primitive function "satisfied" which represents a predicate whose value indicates whether or not the user wishes to continue refining his solution. It might be evaluated by printing the arguments at a terminal and receiving a human response. Other primitive functions in the specifications can be elaborated to provide user interface features (2a)-(2c) of Section 3.1.

As outlined in Chapter 2, *we defined our domain as a union of subdomains* for the expressed purpose of allowing for a natural process decomposition along geometric lines. Since closed subdomains intersect and hence introduce redundant data, we are led to work exclusively with open subdomains. Accordingly a separate process is associated with each one of the 2-, 1-, and 0-subdomains defined before. Figure 3.1 shows these processes for a simple (straightlined) domain.

Each of these processes contains in its state all the information relevant to that particular subdomain. This includes the bilinear form,

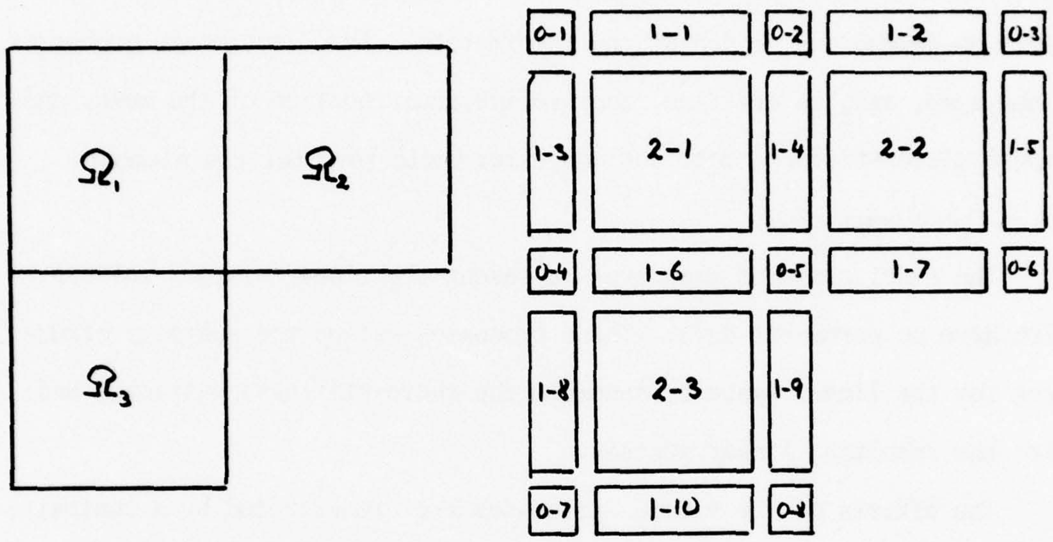

Figure 3.1

load functional, mapping between the base figure and the subdomain, boundary conditions where they apply, as well as the current mesh data and associated solution. In keeping with the performance goal (4) of 3.1 the only data-set stored more than once is the adjacency relation between the subdomains; clearly this is not a large set.

The successor functions of the geometric processes perform all the necessary numerical computations except for the assembly and solution of the resulting linear systems of equations. In particular 0- and 1-subdomain processes answer questions from other processes about their local data, update themselves with new nodal points, new solution values, etc., and report on demand any needed status information. The 2-subdomain processes do the same, and, in addition, they refine their portion of the mesh, and compute micro-stiffness matrices and error indicators for the elements within their boundaries.

The final class of processes represent the linear systems solvers which have no permanent data. These processes set up the sparsity structures for the linear systems, assemble the macro-stiffness matrices, and solve the resulting linear systems.

The efforts of the various processes are orchestrated by a central control process. It receives commands from the current user, gives commands to the geometric processes and the systems solvers, receives reports from them, and sends reports back to the user.

## 3.3 General Systems Operation

As mentioned in Section 3.1, each instance of the system is initialized with a particular domain, bilinear form, and associated error norms. A potential user has to wait until the system is not in use; then he gains exclusive access to it. The user then distributes his specific boundary conditions, load functional, mesh, and associated solutions to the appropriate geometric processes and his status record to the control process. If this is the user's first turn, then the mesh supplied has no solution data and the status record is "empty".

After these initializations the user gives a command to the control process, which in turn executes it and returns a report to him. This is repeated until the user is satisfied, whereupon he receives an updated status record from the control process and gets his specific data, updated mesh, and associated solution back from the geometric processes. He may then begin a post-processing phase or return to inactivity.

Basically a user's command tells the system to refine the current mesh and to solve the resulting (linear) system(s) so as to achieve a prescribed maximal error on the elements without exceeding a given cost. In this section we give an overview of the principal phases of the execution of a command and postpone to the next section a discussion of the cost definition and of the details of the various control decisions.

Generally the execution of a command may consist of one or several solution passes as defined by (2.10). After each solution pass the control process updates a status record containing the data needed for the decisions involved in continued execution or termination. This is the status record received by the user at the end of his turn.

A solution pass begins with the decision about the cutoff values $\varepsilon_{high}$ and $\varepsilon_{low}$ by the control process. The user can set a flag in the status record to override this automatic decision procedure, either by invoking a different user-specified decision process or by deciding himself from a terminal.

After $\varepsilon_{high}$ and $\varepsilon_{low}$ have been determined, these values are sent to all 2-subdomains. This begins the refinement phase. The 2-subdomains proceed to refine their individual meshes in parallel. In general this entails queries and update messages to the adjacent 0- and 1-subdomains. When its refinement is completed, each 2-subdomain process sends to the control process a report listing all the nodes where a new solution is to be computed (active nodes) and giving the total number of elements to which they belong (active elements) as well as the number of refined elements. When all these reports have been obtained, the control process requests all 0- and 1-subdomains to determine from the updates received which of their nodal points are active. The refinement phase is completed when all geometric processes have sent their reports.

Now begins the solution phase. The control process determines the disjoint subsets of nodes where a new solution is to be found. For details of this decision we refer to the next section. The list of nodal points belonging to any such subset is sent to a specific solution process, and the identity of that solution process is communicated to the 2-subdomains that have to supply the corresponding micro-stiffness matrices. From then on several activities proceed in parallel. The necessary micro-stiffness matrices are computed within the various 2-subdomain processes, which in turn query neighboring 0- and 1-subdomains for all necessary information. The solution processes set up and assemble their particular (independent) blocks of the macro-stiffness matrix, incorporating the micro-stiffness matrices from any 2-subdomain whenever they are ready. When such a block assembly is complete, the corresponding linear system is solved and the solutions are sent back to the geometric processes which own the corresponding nodes. When a 2-subdomain process receives a new solution, it calculates new indicators for its elements, estimates the values that may be expected if the elements were to be refined, and sends a report about the results to the control process. All this parallel activity becomes resynchronized when the control process has received a report from all active 2-subdomains and used it to update the status record. This ends the solution phase and with it the solution pass. It should be observed that the parallel computation of the micro-stiffness matrices and of the error calculations provide opportunities for inter- and intra-process parallelism.

## 3.4 Control Decisions

In general user commands are of the form (desired error bound, cost limit, limit on the number of solution passes) and correspondingly the return reports have the form (achieved error bound, cost used, number of solution passes used, total cost used so far during the turn).

The cost expended so far in executing a command is the accumulation of the costs of all the previous solution passes. The cost of a solution pass is known and recorded at the end of the refinement phase when the control process has a report from each subdomain process containing a list of its active nodes and, from each 2-subdomain, the numbers of refined and active elements. Let $r$ and $e$ be the total numbers of refined and active elements respectively and $n_i$ the number of active nodes in solution subset $i$. Then the cost is approximated by

(3.1) $$c_1 r + c_2 e + \sum_i (c_3 n_i + c_4 n_i^3) + c_5 e \; .$$

The constants $c_1, \ldots, c_5$ are implementation-dependent. Some comments about their determination are given in Chapter 4. Generally $c_1$ is the unit cost of refining an element, $c_2$ the unit cost of computing a micro-stiffness matrix, $c_3$ the cost per node of setting up a linear system, $c_4^{1/3}$ the cost per node of solving a linear system, and $c_5$ the unit cost of calculating the error indicator for an element. For later use the ratio

(3.2) $$d = (\sum_i n_i)/e$$

is also recorded at this time.

At the end of the solution phase each 2-subdomain process sends to the control process its achieved error bound, recommendations for high and low cutoffs, and a list of pairs $(\varepsilon_j, e_j)$. The recommended high cutoffs are computed according to the algorithm given in [8] and for the low cutoffs a natural modification of that algorithm is used. The mentioned pairs $(\varepsilon_j, e_j)$ consist of the number $e_j$ of elements in this 2-subdomain with error indicators not less than $\varepsilon_j$. Here the $\varepsilon_j$ are numbers selected a priori, including zero. This list of pairs allows the control process to estimate the effect of a cutoff decision without knowing the error indicators of every element.

The control process now determines the values $\varepsilon_{high}$ and $\varepsilon_{low}$ as the maxima of those recommended by the 2-subdomains. From the list of pairs $(\varepsilon_j, e_j)$ it then estimates the number $r'$ of elements to be refined and the number $e'$ of active elements. With these the cost of the potential solution pass is approximated by

(3.3) $$c_1 r' + c_2 e' + c_3 de' + c_4 (de')^3 + c_5 e'$$

where $c_1, \ldots, c_5$ are the same coefficients as in (3.1) and $d$ is the ratio (3.2). In general this overestimates the cost since it is assumed that the macro-stiffness matrix will not be decomposed into independent blocks. The use of the ratio $d$ to estimate the matrix is based on the assumption that this ratio does not change too much for a given problem.

If the estimated cost (3.3) exceeds the difference between the cost limit and the cost used so far, the execution of the command is terminated. The same is true when the computed a-posteriori estimate falls below the desired error bound. As mentioned before, the user may override these decisions. In particular he may order a recomputation of the solution on all nodal points without any further mesh refinement by setting $\varepsilon_{high} = \infty$, $\varepsilon_{low} = 0$.

The algorithm for the decomposition of the solution subset is based on the adjacency relation between subdomains. Active nodes are placed into the same solution subsets if they belong to two adjacent (open) subdomains. For example, in the case of Figure 3.2 there will be exactly two solution subsets consisting of the active nodes in the subdomain unions (1-1) $\cup$ (2-1) and (2-2) $\cup$ (2-3) $\cup$ (1-3) $\cup$ (1-4) $\cup$ (1-5) $\cup$ (0-1), respectively.

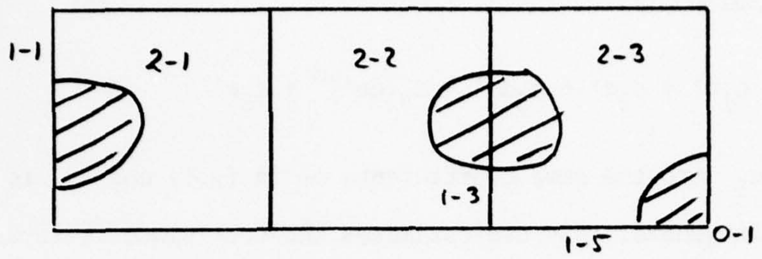

Figure 3.2

## 4. The FEARS Experiment

### 4.1 Implementation

The design described here will be implemented, not as a production system, but as an experimental system for the evaluation of the various new ideas in it, in particular the adaptivity and parallelism approaches. The evaluation techniques will be described in Sections 4.2 and 4.3.

The FEARS (Finite Element, Adaptive Research Solver) system will be a FORTRAN program running on a UNIVAC 1108 under EXEC 8. It will simulate parallel processing and use the process state segmentation as its sole storage management mechanism. It will be structured as an interpreter of the formal specification of this design, calling upon numerical routines as primitive functions.

Our schedule calls for the completion of implementation in Spring 1978 and experimentation during the Summer of 1978.

### 4.2 Evaluation of Adaptivity

It is impossible to quantify the convenience to the user of an adaptive system. All we can measure is the relative costs of adaptive and nonadaptive solutions.

The adaptive and nonadaptive cost of executing each command will be recorded. The adaptive cost is measured as described in Section 3.4. The nonadaptive cost is understood to be the cost, as computed by (3.1), of one solution pass for a uniform mesh with elements the size of the smallest elements used in the adaptive solution.

As mentioned earlier, the constants $c_i$ in (3.1) and (3.3) are implementation-dependent, but their ratios should be much the same for any implementation. They will be approximated for our purposes by timing representative code on an otherwise idle machine. The overhead of inter-process communication will not be included, as that is considered in the evaluation of parallelism.

## 4.3 Evaluation of Parallelism

Without measurement we cannot show that the performance goals (1) and (3) of Section 3.1 have been met, that is, the goals of effective parallelism and useful data segmentation. It is also necessary to see that the benefits of parallel structure are not outweighed by the attendant interprocess communication overhead.

The measure of parallelism will be a "speedup factor," which quantifies how many times faster an implementation with an arbitrarily large number of processors could finish a problem than an implementation with a single processor of the same speed. It cannot be determined by sampling techniques because analytical results do not apply to parallel activities with such a high degree of interdependence. Therefore we will treat each "run" of a process, which is uninterrupted by the need to communicate, as a separate task and record how long it took and which other tasks it enabled to run when it stopped in order to communicate. This information will later be fed through a simple infinite-resource simulator to determine the average speedup.

Data segmentation will be measured by the numbers of elements and nodes in subdomain processes and the numbers of points in linear systems. These are good implementation-independent indicators of relative sizes of the data structures.

Communication overhead will be measured simply by counting the number of communications which take place.

These implementation-independent measurements will be taken separately over the refinement and solution phases of every solution pass. For proper evaluation it will be necessary to relate them to problem-dependent factors. The observed speedup factor, for instance, is highly dependent on the number of 2-subdomains. Therefore the post-execution processor of this data will provide roughly calculated "efficiency factors" relating observed and ideal quantities. The output is summarized in Table 4.

The final evaluation of these numbers, of course, cannot be done independently of an implementation. For instance, more efficient communication mechanisms make a higher overhead rate feasible.

## Acknowledgments

We are pleased to acknowledge the participation of Professor Ivo Babuška and Mr. Charles Mesztenyi in the planning and development of the FEARS project.

| Phase of Solution Pass | Output Factor | Purpose of Output Factor |
|---|---|---|
| | speedup | measure of effective parallelism |
| | number of communications | measure of overhead |
| | numbers of nodes and elements in all 2-subdomains | measure of data segmentation |
| refinement phase | number of 2-subdomains | used for comparisons |
| | speedup/number of 2-subdomains | efficiency factor for parallelism |
| | number of communications/ total number of elements | efficiency factor for overhead |
| | speedup | measure of effective parallelism |
| | number of communications | measure of overhead |
| | numbers of points in all linear systems | measure of data segmentation |
| solution phase | number of 2-subdomains | used for comparison |
| | number of linear systems | used for comparison |
| | speedup/[(number of 2-subdomains)+(number of linear systems)/2] | efficiency factor for parallelism |
| | number of communications/ total number of points | efficiency factor for overhead |

Table 4

## References

1. W. Pilkey, K. Saczalski, H. Schaeffer (editors), Structural Mechanics Computer Programs, Survey, Assessments and Availability, University of Virginia Press, Charlottesville, Va., 1974.

2. R. H. Gallagher, Computerized structural analysis and design--the next twenty years, Keynote Lecture, Second National Symposium on Computerized Structural Analysis, George Washington University, Washington, D.C., 1976.

3. E. Schrem, Finite element software in the next decade, Proceedings World Congress on the Finite Element Method, Bournemouth, England, 1975.

4. I. Babuška, W. Rheinboldt, C. Mesztenyi, Self-adaptive refinement in the finite element method, University of Maryland, Computer Science Technical Report TR-375, 1975.

5. I. Babuška, The self-adaptive approach in the finite element method, The Mathematics of Finite Elements and Applications II, MAFELAP 1975, J. R. Whiteman (editor), Academic Press, New York, 1976, 125-142.

6. I. Babuška, W. Rheinboldt, Mathematical problems of computational decisions in the finite element method, Mathematical Aspects of Finite Element Methods, I. Galligani, E. Magenes (editors), Lecture Notes in Mathematics, Vol. 606, Springer Verlag/Germany, 1977, 1-26.

7. I. Babuška, W. Rheinboldt, Computational aspects of finite element analysis, Mathematical Software--III, J. R. Rice (editor), Academic Press, New York, 223-253, in press.

8. I. Babuška, W. Rheinboldt, Error estimates for adaptive finite element computations, SIAM J. Num. Anal., in press.

9. I. Babuška, W. Rheinboldt, A-posteriori error estimates for the finite element method, Int. J. Num. Meth. in Eng., submitted for publication.

10. D. R. Fitzwater, The formal design and analysis of distributed data-processing systems, University of Wisconsin, Computer Sciences Department, TR-295, 1977.

11. Pamela Zave, D. R. Fitzwater, Specification of asynchronous interactions using primitive functions, submitted for publication.

12. D. R. Fitzwater, Pamela Zave, The use of formal asynchronous process specifications in a system development process, Sixth Texas Conference on Computing Systems, 1977.

13. I. Babuška, A. K. Aziz, Survey lectures on the mathematical foundations of the finite element method, The Mathematical Foundations of the Finite Element Method with Applications to Partial Differential Equations, A. K. Aziz (editor), Academic Press, New York, 1972, 3-359.

14. K. J. Bathe, E. L. Wilson, Numerical Methods in Finite Element Analysis, Prentice-Hall, Englewood Cliffs, N.J., 1976.

15. W. J. Gordon, Blending-function methods for bivariate and multivariate interpolation and approximation, SIAM J. Num. Anal. 8, 1971, 158-177.

## Appendix: Detailed Process Descriptions

In the actual system the numbers of all types of processes are bounded, and the full set of processes exists in a static structure even when some are not in use. The processes which are being used are called the "busy" processes.

For readability the actions of successor functions will be described in a pseudo-programming language. Comments are in square brackets.

### User Processes

Named U0 through $Ub_u$, where $b_u$ is one less than the bound on the number of users.

State contains:

(1) phase ( is this the initialization step or a subsequent step?);

(2) adjacency relation on existing subdomains;

(3) status record;

(4) load (the boundary conditions and load functional for this user, divided and indexed according to subdomains);

(5) mesh (the current mesh for this user, with solutions, divided and indexed according to subdomains);

(6) whatever local data is needed for pre- or post-processing.

Successor function:

```
begin [Step of user process.]
  if "this is initialization step" then
    begin [Initialization.]
      "get adjacency relation";
      [U0  is special because it is the process that creates the domain.
      Thus for  U0  this operation entails creating the domain and bilinear
      form (assigning numbers to all subdomains), sending the pieces of the
      domain and bilinear form to the appropriate geometric processes, and
      saving a copy only of the adjacency relation on the existing sub-
      domains.  U0  keeps one copy of this relation, sending one copy to
      the control process and one to every other user process.  For
      $U_j$, $1 \leq j \leq b_u$, this operation entails only receiving this informa-
      tion from  U0.]
      "create initial load and mesh"
      [This operation can take as long as is necessary for a jth user to
      arrive on the scene.  It can make use of the adjacencies directly,
      or the bilinear form by querying the appropriate geometric processes.
      This operation (and the preceding one) represent pre-processing and
      can be arbitrarily complex.]
    end [Initialization.]
  else
    begin [Processing step.]
      "get control of finite-element system and send a copy of status record
      to it";
      "send load and mesh pieces to appropriate busy geometric processes";
      signal := go;
      report := none;
      for i := 1 step 1 until  $b_c$  do
      [The bound on the number of commands a user may give is  $b_c$.]
        begin [User's turn.]
          if signal = go then
            begin
              if "satisfied with report" then
                begin
                  "command control to stop";
                  signal := stop
                end
              else
                begin
                  "make a command, referring to report and old status records";
                  "send command to control";
                  report := "received report"
                end
            end
        end [User's turn.]
      "get load and mesh pieces back from busy geometric processes";
      "get updated copy of status record back and give up control of system";
      "post-process"
      [During post-processing, the user may get information about the bilinear
      form by querying subdomain processes.]
    end [Processing step.]
end [Step of user process.]
```

The Control Process

State contains:

(1) phase (is this the initialization step or a subsequent step?);

(2) adjacency relation on subdomains.

Successor function:

```
begin [Step of control process.]
  if "this is initialization step" then
    "get adjacency relation"
  else
    begin [Processing step.]
      status := "received user status";
      [Getting the control process to accept its status record is how a
      user gets exclusive access.]
      signal := go;
      for i := 1 step 1 until $b_c$ do
      [The bound on the number of commands a user may give is $b_c$.]
        begin [One user's turn.]
          if signal = go then
          begin
            command := "received command";
            if command = stop then
              signal := stop
              else
                begin [Command execution.]
                [This code appears subsequently.]
                end [Command execution.];
              "return report to user"
            end
          end [One user's turn.]
        "return updated status to user";
      end [Processing step.]
end [Step of control process.]

begin [Command execution.]
  inner-signal := go;
  for k := 1 step 1 until  $b_T$  do
  [The bound on the number of solution passes per command is  $b_T$.]
    if inner-signal = go then
      begin
        if "command execution finished" then
        [This is decided by an algorithm whose arguments are the command
        and the status record, which contains the results of the last
        solution pass.  Alternatively, it can be decided by asking a user
        supplied decision process, which does nothing but answer such
        questions and is not explicitly described here.]
          inner-signal := stop
        else
          begin [Solution pass.]
            "decide cutoffs";
            [As above, this decision can be done manually or automatically
            and can be based on the same information.]
            "command all busy 2-subdomain processes to refine to cutoffs";
            "receive answers from all busy 2-subdomain processes";
            [Each answer consists of a list of points to be included in the
            solution, plus data needed to calculate the costs of refinement
            and solution.]
            "command all busy 0- and 1-subdomain processes to report";
            [By this time these processes know which of their points are
            to be included in the solution  and send them to the control
            process.]
            "receive answers from all busy 0- and 1-subdomain processes";
            "calculate solution subsets";
            "record costs of this solution pass in status record";
            [At this time all cost factors (number of elements refined,
            sizes of linear systems, number of elements included in solution)
            are known.]
            "send point lists to active solution processes and solution
            process names to 2-subdomain processes included in solution";
            "receive reports from all 2-subdomain processes included in
            solution";
            "update status record with new error information"
        end [Solution pass.]
    end
end [Command execution.]
```

2-Subdomain Processes

Named S1 through $Sb_s$, where $b_s$ is the bound on the number of 2-subdomains in the domain.

State contains:

(1) domain (subdomains adjacent to this one, mapping functions, appropriate piece of bilinear form);

(2) load (appropriate pieces of user-dependent information: boundary conditions and load functional);

(3) mesh (appropriate piece of user-dependent mesh, with solutions).

Successor function:

```
begin [Step of 2-subdomain process.]
  "get command";
  [At the beginning of each step, this process waits until it gets a command
  to execute.  Each command consists of a type and some data.]
```

```
if "command to initialize domain" then
[This command comes from UO.]
  "set domain structure to value of data"
else if "command to initialize user" then
[This command comes from a user process.]
  "set load and mesh structure to value of data"
else if "command to finalize user" then
[This command comes from a user process.]
  "send load and mesh back"
  [Since there will only be one user waiting for this information, the
  identity of the particular user need not be known by this process.]
else if "command is a query" then
[A query can come from any user doing pre- or post-processing, or another
geometric process.]
  begin
    "formulate answer";
    [A query can ask anything about the local data of this process.]
    "send answer"
    [Since only one questioner is waiting for an answer, its identity need
    not be known.]
  end
else if "command is to refine" then
[This command comes only from the control process.]
  begin
    "refine elements above high error cutoff";
    [This may entail updating adjacent 1-subdomains with new points.]
    "collect solution information";
    [Including all new elements and all those above the low error cutoff,
    make a list of all active nodes, plus a tally of refined and active
    elements.]
    "send solution information to control"
  end
else if "command is to solve" then
[This command comes only from the control process.]
```

```
    begin
      "compute micro-stiffness matrices and right-hand sides for all active
      elements";
      [These can be done in parallel and may entail queries to adjacent
      0- and 1-subdomain processes.]
      "send to solution process"
      [The identification of the solution process is the data.]
    end
  else
  [The only remaining kind of command comes from a solution process and
  brings new solutions.]
    begin
      "put new solutions in mesh";
      "calculate new error indicators on active elements";
      [These can be done in parallel.]
      "formulate a report";
      "send report to control process"
    end
end [Step of 2-subdomain process.]
```

0- and 1-Subdomain Processes

Named P1 through $Pb_p$ and L1 through $Lb_\ell$, where $b_p$ and $b_\ell$ are the bounds on the numbers of 0- and 1-subdomains, respectively.

State contains:

(1) domain (subdomains adjacent to this one, mapping functions, appropriate pieces of bilinear form for 1-subdomain processes);

(2) load (boundary conditions and load functional);

(3) mesh (appropriate pieces of user-dependent mesh with solutions).

Successor function:

```
begin [Step of 0- or 1-subdomain process.]
  "get command";
  [The commands to initialize the domain, initialize a user, finalize a
  user, or answer a query, are identical to those for 2-subdomains and
  will not be repeated here.]
  if "command is to update" then
  [This command can only come from another subdomain process.]
    "perform requested update of local data"
  else if "command is to report" then
  [This command comes only from the control process.]
    begin
      "collect solution information and reset";
      [The solution information is the list of points to be included in
      the solution.  Since this decision can only be made as a result of
      queries and updates from 2-subdomain processes during refinement,
      that information is kept during refinement and reset here (so that
      a different set of nodes can be active in the next solution pass).]
      "send to control process"
    end
  else
  [The only remaining command brings new solutions from a solution process.]
    "put new solutions in mesh"
end [Step of 0- or 1-subdomain process.]
```

Solution Processes

Named V1 through $Vb_V$, where $b_V$ is the bound on the number of simultaneous solutions.

State contains: nothing.

Successor function:

```
begin [Step of solution process.]
  "get point list from control process";
  "make template of linear system";
  for i := 1 step 1 until "number of 2-subdomains involved" do
    begin
      "get micro-stiffness matrices and right-hand sides from a 2-subdomain
      process";
      [The 2-subdomain process can be any one of those assigned to this
      solution process, presumably the one that is ready next.]
      "assemble list into linear system"
    end;
  "solve linear system";
  "send solutions to appropriate geometric processes"
  [Each point has a unique name which includes the subdomain to which it
  belongs.]
end [Step of solution process.]
```

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

**REPORT DOCUMENTATION PAGE** — READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: ONR-N00014-77-C-0623-593
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER:
4. TITLE (and Subtitle): 6 Design of an Adaptive, Parallel Finite Element System.
5. TYPE OF REPORT & PERIOD COVERED: 9 Technical Rep[redacted]t.,
6. PERFORMING ORG. REPORT NUMBER: 14 TR-593
7. AUTHOR(s): 10 Pamela Zave [redacted] Werner C. Rheinboldt
8. CONTRACT OR GRANT NUMBER(s): 15 N00014-77-C-0623
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Computer Science Center, University of Maryland, College Park, MD 20742
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research, Mathematics Branch, Arlington, VA 22217
12. REPORT DATE: 11 Nov[redacted]77
13. NUMBER OF PAGES: ~~42~~ 12 44p.
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):
15. SECURITY CLASS. (of this report): UNCLASSIFIED
15a. DECLASSIFICATION DOWNGRADING SCHEDULE:
16. DISTRIBUTION STATEMENT (of this Report): Approved for public release; distribution unlimited.
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report):
18. SUPPLEMENTARY NOTES:
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

linear, elliptic boundary value problems
finite element software
adaptive error estimation
parallel process structure

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A design for a novel prototype finite element system is presented which meets the following four goals: (a) The system constitutes an application-independent finite element solver for a certain class of linear elliptic problems based on a weak mathematical formulation; (b) it incorporates extensive adaptive approaches to minimize the critical decisions demanded of the user; (c) it incorporates adaptive error estimation techniques to provide an optimal solution within a prescribed cost range; and ~~(last but not least)~~ (d) it takes → Don't Req

[illegible]

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

403018 UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. Abstract (continued)

advantage of natural parallelism and modularity to increase the size of the practically solvable problems. The overall system structure is described in terms of parallel processes and is implementable on a wide variety of hardware configurations. The experiments planned to evaluate the various new approaches are also presented.